# Dynamic Programming in RL

**Dynamic programming** algorithms solve a category of problems called planning problems. Herein given the complete model and specifications of the environment (MDP). Dynamic Programming methods are powerful for solving RL problems optimally and are typically used when we know the transition Dynamics and reward function for the environment we are trying to solve for.

The two required properties of dynamic programming are:

- ***Optimal substructure***: optimal solution of the sub-problem can be used to solve the overall problem.
- ***Overlapping subproblems***: sub-problems recur many times. Solutions of sub-problems can be cached and reused

Markov Decision Processes satisfy both of these properties:

- *The Bellman Equation gives recursive decomposition* which tells us how to break down the optimal value function into two pieces. i.e the optimal behaviour of one step and then the optimal value of the remaining steps.
- *The Value function stores and reuses solutions*. Cache with all the good information of the MDP which tells you the optimal reward you can get from that state onward.

# Dynamic Programming methods:

**Policy Iteration:**

This Dynamic Programming method mainly Consist of two steps:

1. **Policy Evaluation :** Policy evaluation is the process of estimating the value function for a given policy. It can be done iteratively using the Bellman equation.
2. **Policy Improvement :** involves updating the current policy to maximize the expected return according to the current value function.

   **Approach** :
   1. We first start with a completely random policy, then by Iterating over Bellman equation until convergence of the value function for current policy

$$v_{k+1}(s) = \sum_{a \in A} \pi(a|s) \left( R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_k(s') \right)$$

Bellman Expectation Equation

In simpler terms : **"The expected total reward (V) the agent can get from a state (s) is equal to the immediate reward it receives in that state plus the expected total reward it can get from the next state (s') if it continues to follow its policy."**

2. Next we update our current policy using the estimated by function by acting greedy and updating our policy with step/actions which **maximizes the our action value function**, Basically the agent selects the action in each state that maximizes the expected cumulative reward according to the current value function.

$$\pi' = argmax(Q(s, a))$$

- π'(s) represents the improved policy for state s.
- argmax(Q(s, a)) finds the action that yields the highest Q-value for that state.

We repeatedly perform policy evaluation and improvement steps till we converge to an optimal policy.

**Value Iteration:**
The value Iteration method can be simply thought of as one single step of policy evaluation( value function estimation using bellman optimality equation) directly followed by a policy improvement step repeatedly until the convergence of the value function.

**Approach:**

1. Begin by initializing the value function for all states arbitrarily, typically to zero or some small random values.
2. In each iteration, update the value of each state based on the Bellman optimality equation.

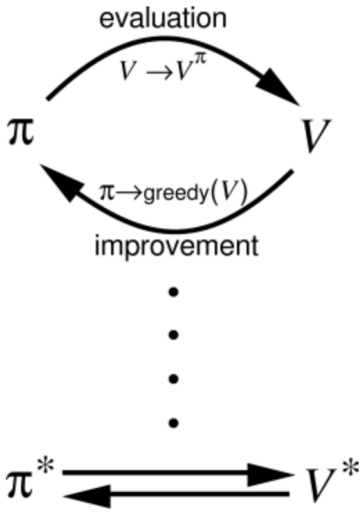
$$Q(s) = max(\Sigma[R(s, a, s') + \gamma * V(s')])$$

3. Repeat the iterative updates until the change in the value function between successive iterations is below a certain threshold or until it converges. The value function converges to the optimal value function, and the policy converges to the optimal policy.
4. After the convergence, you can extract the optimal policy by selecting the action in each state that maximizes the right-hand side of the Bellman optimality equation. This policy is guaranteed to be an optimal policy for the MDP.

In practice, Policy iteration converges faster than value iteration but policy iteration is more computationally expensive than value iteration and may perform poorer in large state-action space.

**Generalized Policy Iteration:**

the term *generalized policy iteration* (GPI) to refer to the general idea of letting policy evaluation and policy improvement processes interact. All the RL learning methods have identifiable policies and value functions, with the policy always being improved with respect to the value function and the value function always being driven toward the value function for the policy.




The evaluation and improvement processes in GPI can be viewed as both competing and cooperating. They compete in the sense that they pull in opposing directions. Making the policy greedy with respect to the value function typically makes the value function incorrect for the changed policy, and making the value function consistent with the policy typically causes that policy no longer to be greedy. In the long run, however, these two processes interact to find a single joint solution: the optimal value function and an optimal policy.

if both the evaluation process and the improvement process stabilize, that is, no longer produce changes, then the value function and policy must be optimal.

**Pros and Cons Dynamic Programming methods:**

**Pros:**

1. DP methods, such as Policy Iteration and Value Iteration, guarantee convergence to an optimal policy when used on finite Markov Decision Processes (MDPs).
2. DP provides a clear and mathematically sound framework for solving RL problems. Mostly driven by greedy nature of problems
3. DP methods can be very efficient when the MDP model is known because they take advantage of this knowledge to compute optimal policies efficiently.
4. DP methods typically converge to the optimal solution over time, ensuring that the solution will improve with more iterations.

**Cons:**

1. DP methods are infeasible for large and high-dimensional problems due to high space and time complexity.
2. DP methods assume complete knowledge of the environment, which is often an unrealistic assumption in real-world applications. Many real problems involve partial observability, unknown dynamics, or noisy observations.